CS 294-4 IRAM: Intelligent DRAM
Lecture 3: Cache Review
1/22/95

Dave Patterson

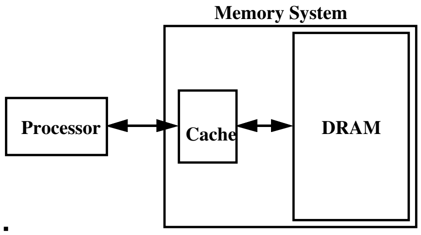
Computer Science Division
University of California, Berkeley

`http://www.cs.berkeley.edu/~patterson/`

Outline

- **Technological motivation for cache**
- **Cache Terms**
- **Styles of Caches**
- **Categorzing Cache Misses**
- **Memory Organizations**
- **Advanced Topics (as time permits): non-blocking cache, 2 level cache, sub-block, prefetching, SW opt.**

The Motivation for Caches

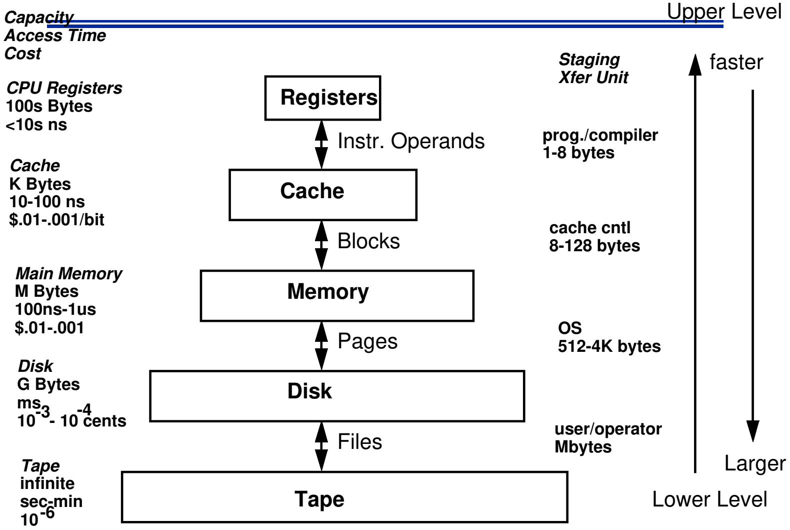


- **Motivation:**
  - Large memories (DRAM) are slow
  - Small memories (SRAM) are fast
- **Make the *average access time* small by:**
  - Servicing most accesses from a small, fast memory.
- **Reduce the *bandwidth* required of the large memory**

The Principle of Locality

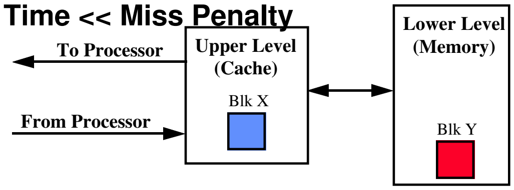
- **The Principle of Locality:**
  - Program access a relatively small portion of the address space at any instant of time.
  - Example: 90% of time in 10% of the code
- **Two Different Types of Locality:**
  - *Temporal Locality* (Locality in Time): If an item is referenced, it will tend to be referenced again soon.
  - *Spatial Locality* (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon.

Levels of the Memory Hierarchy



Memory Hierarchy: Terminology

- ***Hit***: data appears in some block in the upper level (example: Block X)
  - ***Hit Rate***: the fraction of memory access found in the upper level
  - ***Hit Time***: Time to access the upper level which consists of
    RAM access time + Time to determine hit/miss
- ***Miss***: data needs to be retrieve from a block in the lower level (Block Y)
  - ***Miss Rate*** = 1 - (Hit Rate)
  - ***Miss Penalty*** = Time to replace a block in the upper level +
    Time to deliver the block the processor
- Hit Time << Miss Penalty



Cache Measures

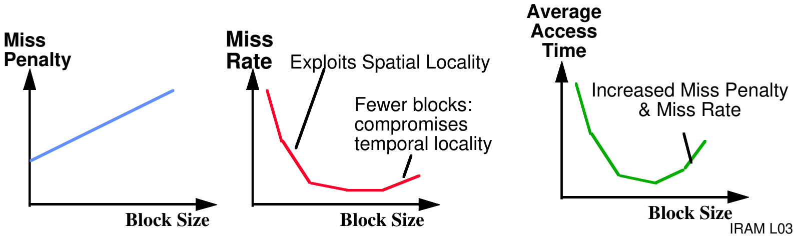
- *Hit rate*: fraction found in that level
  - So high that usually talk about *Miss rate*
  - Miss rate fallacy: as MIPS to CPU performance, miss rate to average memory access time in memory
- Average memory-access time
  = Hit time + Miss rate x Miss penalty (ns or clocks)
- *Miss penalty*: time to replace a block from lower level, including time to replace in CPU
  - *access time*: time to lower level
    = f(lower level latency)
  - *transfer time*: time to transfer block
    =f(BW upper & lower)

Improving Cache Performance

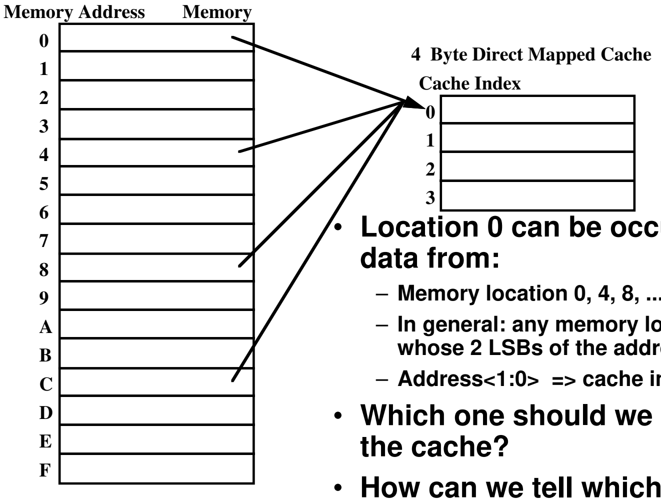
1. Reduce the miss rate,
2. Reduce the miss penalty, or
3. Reduce the time to hit in the cache.

Block Size Tradeoff

- In general, larger block size take advantage of spatial locality BUT:
  - Larger block size means larger miss penalty:
    - Takes longer time to fill up the block
  - If block size is too big relative to cache size, miss rate will go up
- Average Access Time:
  - = Hit Time x (1 - Miss Rate) + Miss Penalty x Miss Rate



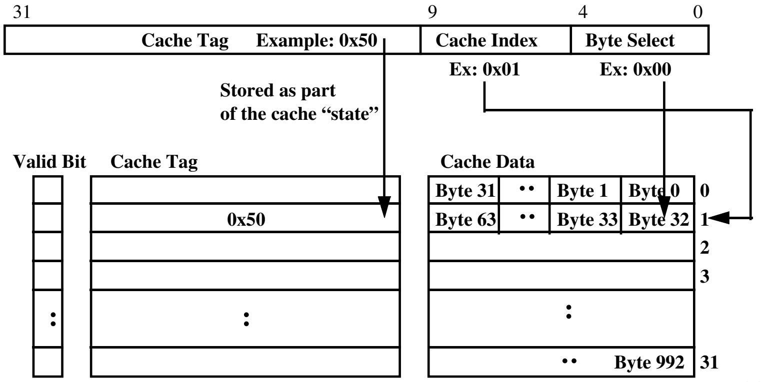
The Simplest Cache: Direct Mapped Cache



- Location 0 can be occupied by data from:
  - Memory location 0, 4, 8, ... etc.
  - In general: any memory location whose 2 LSBs of the address are 0s
  - Address<1:0> => cache index
- Which one should we place in the cache?
- How can we tell which one is in the cache?

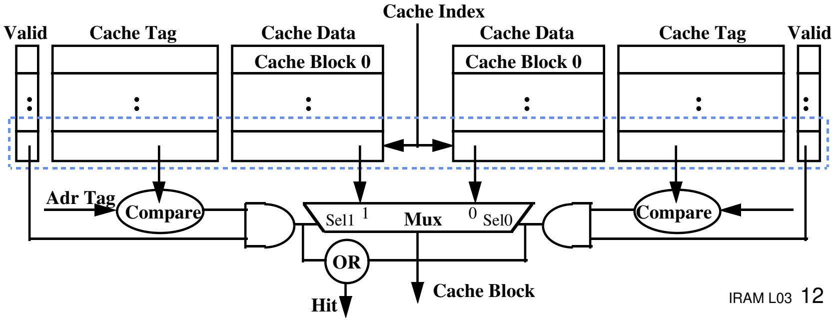
Example: 1 KB Direct Mapped Cache with 32 B Blocks

- For a 2 ** N byte cache:
  - The uppermost (32 - N) bits are always the Cache Tag
  - The lowest M bits are the Byte Select (Block Size = 2 ** M)



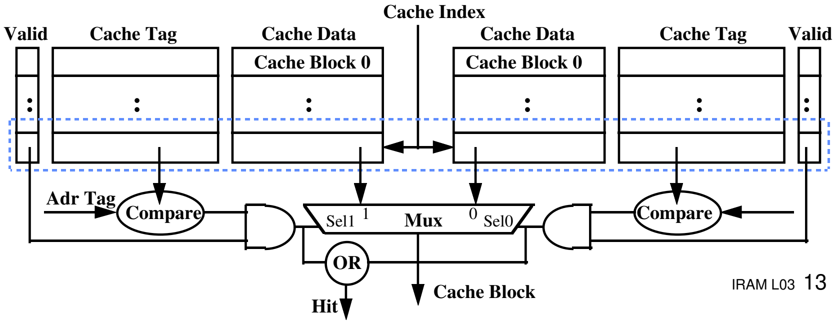
A Two-way Set Associative Cache

- N-way set associative: N entries for each Cache Index
  - N direct mapped caches operates in parallel
- Example: Two-way set associative cache
  - Cache Index selects a "set" from the cache
  - The two tags in the set are compared in parallel
  - Data is selected based on the tag result



Disadvantage of Set Associative Cache

- **N-way Set Associative Cache v. Direct Mapped Cache:**
  - N comparators vs. 1
  - Extra MUX delay for the data
  - Data comes AFTER Hit/Miss
- **In a direct mapped cache, Cache Block is available BEFORE Hit/Miss:**
  - Possible to assume a hit and continue. Recover later if miss.

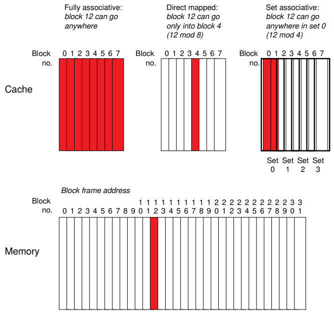


4 Questions for Memory Hierarchy

- **Q1: Where can a block be placed in the upper level?** ***(Block placement)***
- **Q2: How is a block found if it is in the upper level?**
  (Block identification)
- **Q3: Which block should be replaced on a miss?**
  (Block replacement)
- **Q4: What happens on a write?**
  (Write strategy)

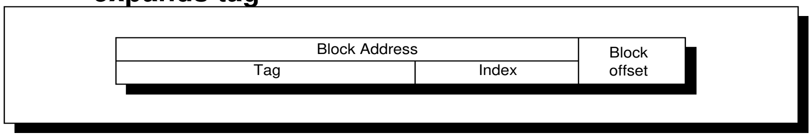
Q1: Where can a block be placed in the upper level?

- **Block 12 placed in 8 block cache:**
  - **Fully associative, direct mapped, 2-way set associative**
  - **S.A. Mapping = Block Number Modulo Number Sets**



Q2: How is a block found if it is in the upper level?

- **Tag on each block**
  - **No need to check index or block offset**
- **Increasing associativity shrinks index, expands tag**



Q3: Which block should be replaced on a miss?

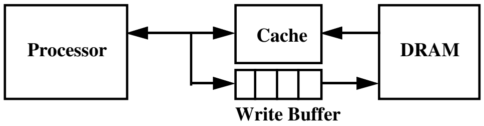
- Easy for Direct Mapped
- S.A. or F.A.:
  - Random
  - LRU (Least Recently Used)

Associativity:	2-way		4-way		8-way	
Size	LRU	Random	LRU	Random	LRU	Random
16 KB	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64 KB	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

Q4: What happens on a write?

- ***Write through***—The information is written to both the block in the cache and to the block in the lower-level memory.
- ***Write back***—The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.
  - is block clean or dirty?
- Pros and Cons of each?
  - WT: read misses cannot result in writes
  - WB: no writes of repeated writes
- WT always combined with write buffers so that don’t wait for lower level memory

Write Buffer for Write Through

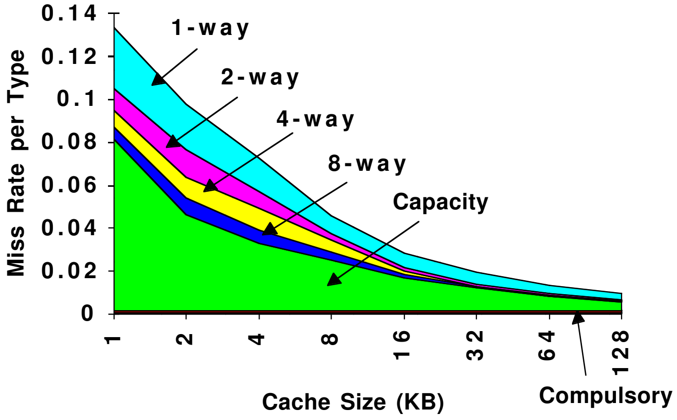


- A Write Buffer is needed between the Cache and Memory
  - Processor: writes data into the cache and the write buffer
  - Memory controller: write contents of the buffer to memory
- Write buffer is just a FIFO:
  - Typical number of entries: 4
  - Works fine if: Store frequency (w.r.t. time) << 1 / DRAM write cycle
- Memory system designer's nightmare:
  - Store frequency (w.r.t. time) -> 1 / DRAM write cycle
  - Write buffer saturation

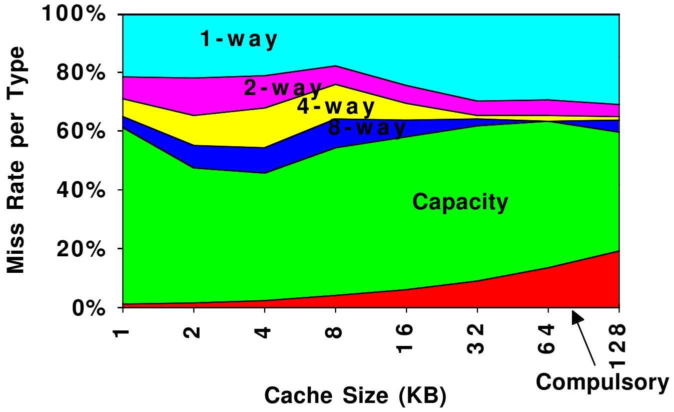
Reducing Misses

- Classifying Misses: 3 Cs
  - *Compulsory*—The first access to a block is not in the cache, so the block must be brought into the cache. These are also called *cold start misses* or *first reference misses*.
    (Misses in Infinite Cache)
  - *Capacity*—If the cache cannot contain all the blocks needed during execution of a program, capacity misses will occur due to blocks being discarded and later retrieved.
    (Misses in Size X Cache)
  - *Conflict*—If the block-placement strategy is set associative or direct mapped, conflict misses (in addition to compulsory and capacity misses) will occur because a block can be discarded and later retrieved if too many blocks map to its set. These are also called *collision misses* or *interference misses*.
    (Misses in N-way Associative, Size X Cache)

3Cs Absolute Miss Rate *(2:1 Cache Rule)*



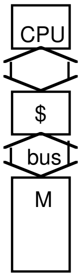
3Cs Relative Miss Rate



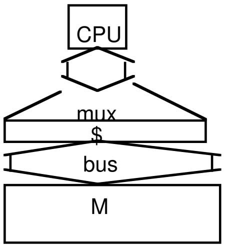
How Can Reduce Misses?

Change Associativity? Which of 3Cs affect?
Change Block Size? Which of 3Cs affect?
Change Compiler? Which of 3Cs affect?

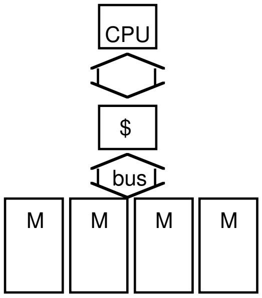
Reducing Memory Transfer Time



Solution 1
High BW DRAM



Solution 2
Wide Path Between Memory & Cache



Solution 3
Memory Interleaving

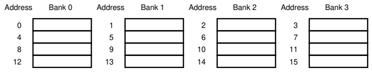
Main Memory Performance

Simple: CPU, Cache, Bus, Memory same width (32 bits)

Wide: CPU/Mux 1 word; Mux/Cache, Bus, Memory N words (Alpha: 64 bits & 256 bits)

Interleaved: CPU, Cache, Bus 1 word: Memory N Modules (4 Modules); example is ***word interleaved***

Timing model: 1 to send address,
6 access time, 1 to send data
Cache Block is 4 words
Simple M.P. **= 4 x (1+6+1) = 32**
Wide M.P. **= 1 + 6 + 1 = 8**
Interleaved M.P. **= 1 + 6 + 4x1 = 11**



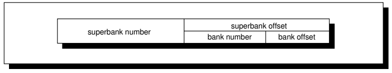
Independent Memory Banks

Memory banks for independent accesses vs. faster sequential accesses

≅ **Multiprocessor**
≅ **I/O**
≅ **Miss under Miss, Non-blocking Cache**

Superbank: **all memory active on one block transfer**

Bank: **portion within a superbank that is word interleaved**



How many banks?

number banks   number clocks to access word in bank

≅ **For sequential accesses, otherwise will return to original bank before it has next word ready**

Increasing DRAM => fewer chips => harder to have banks

Avoiding Bank Conflicts

Lots of banks

```
int x[256][512];
  for (j = 0; j < 512; j = j+1)
     for (i = 0; i < 256; i = i+1)
         x[i][j] = 2 * x[i][j];
```

Even with 128 banks, since 512 is multiple of 128, conflict

SW: loop interchange or declaring array not power of 2

HW: Prime number of banks

- bank number = address mod number of banks
- address within bank = address / number of banks
- modulo & divide per memory access?
- address within bank = address mod number words in bank
- bank number?

Fast Bank Number

Chinese Remainder Theorem: As long as two sets of integers ai and bi follow these rules

$$b_i = x \bmod a_i, 0 \le b_i < a_i, 0 \le x < a_0 \times a_1 \times a_2 \times \ldots$$

and that ai and aj are co-prime if i j, then the integer x has only one solution (unambiguous mapping):

- bank number = b_0, number of banks = a_0
- address within bank = b_1, number of words in bank = a_1
- N word address 0 to N-1, prime no. banks, words power of 2

		Seq. Interleaved			Modulo Interleaved	
Bank Number:	0	1	2	0	1	2
Address within Bank: 0	0	1	2	0	16	8
1	3	4	5	9	1	17
2	6	7	8	18	10	2
3	9	10	11	3	19	11
4	12	13	14	12	4	20
5	15	16	17	21	13	5
6	18	19	20	6	22	14
7	21	22	23	15	7	23

Cache Optimization Summary

Technique	*MR*	*MP*	*HT*	*Complexity*
Larger Block Size	+	≅		0
Higher Associativity	+		≅	1
Victim Caches	+			2
Pseudo-Associative Caches	+			2
HW Prefetching of Instr/Data	+			2
Compiler Controlled Prefetching	+			3
Compiler Reduce Misses	+			0
Priority to Read Misses		+		1
Subblock Placement		+	+	1
Early Restart & Critical Word 1st		+		2
Non-Blocking Caches		+		3
Second Level Caches		+		2
Small & Simple Caches	≅		+	0
Avoiding Address Translation			+	2
Pipelining Writes			+	1

Reduce MP: Non-blocking Caches to reduce stalls on misses

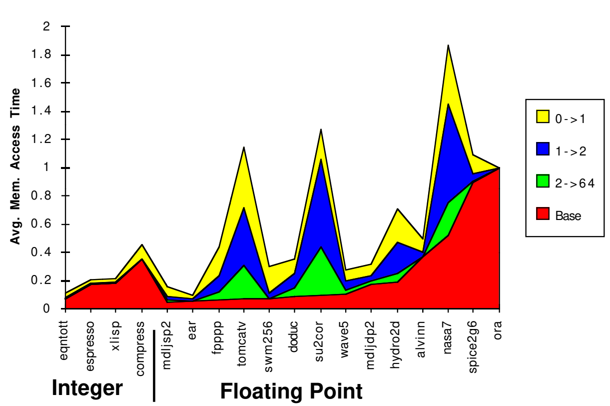
non-blocking cache or *lockup-free cache* allowing the data cache to continue to supply cache hits during a miss

∋*hit under miss* reduces the effective miss penalty by being helpful during a miss instead of ignoring the requests of the CPU

∋*hit under multiple miss* or ∋ *miss under miss* may further lower the effective miss penalty by overlapping multiple misses

- ≅ Significantly increases the complexity of the cache controller as there can be multiple memory accesses

Value of Hit Under Miss for SPEC



FP programs on average: AMAT= 0.68 -> 0.52 -> 0.34 -> 0.26

Int programs on average: AMAT= 0.24 -> 0.20 -> 0.19 -> 0.19

8 KB Data Cache, Direct Mapped, 32B block, 16 cycle miss

Miss Penalty Reduction: Second Level Cache

L2 Equations

$$Average\ \ memory\ \ access\ \ time = Hit\ \ time_{L1} + Miss\ \ rate_{L1} \times Miss\ \ penalty_{L1}$$

$$Miss\ \ penalty_{L1} = Hit\ \ time_{L2} + Miss\ \ rate_{L2} \times Miss\ \ penalty_{L2}$$

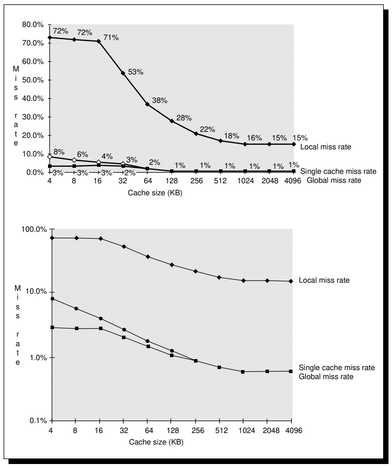
$$Average\ \ memory\ \ access\ \ time = Hit\ \ time_{L1} + Miss\ \ rate_{L1} \times \left(Hit\ \ time_{L2} + Miss\ \ rate_{L2} \times Miss\ \ penalty_{L2}\right)$$

Definitions:

- ***Local miss rate*** **misses in this cache divided by the total number of memory accesses *to this cache* (Miss rate$_{L2}$)**
- ***Global miss rate*** **misses in this cache divided by the total number of memory accesses *generated by the CPU* (Miss rate$_{L1}$ x Miss rate$_{L2}$)**

Comparing Local and Global Miss Rates

- Linear vs. Log scale
- Global miss rate close to single level cache rate provided L2 >> L1
- Don't use local miss rate
- L2 not tied to CPU clock cycle
- Cost & A.M.A.T.
- Generally Fast Hit Times and fewer misses
- Since hits are few, target miss reduction



Reducing Miss Penalty Summary

- 5 techniques
  - Read priority over write on miss
  - Subblock placement
  - Early Restart and Critical Word First on miss
  - Non-blocking Caches (Hit Under Miss)
  - Second Level Cache
- Can be applied recursively to Multilevel Caches
  - Danger is that time to DRAM will grow with multiple levels in between

What is a Sub-block?

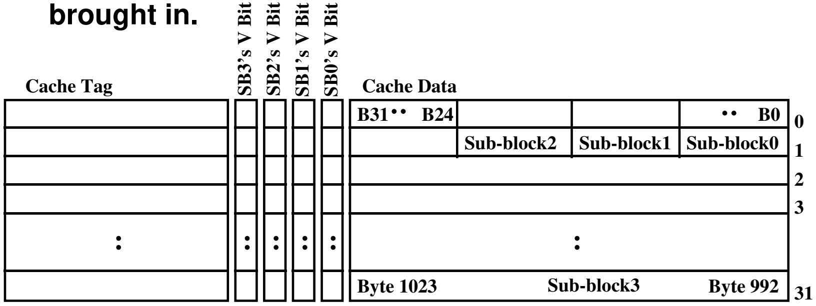
Sub-block:

≅ A unit within a block that has its own valid bit

≅ Example: 1 KB Direct Mapped Cache, 32-B Block, 8-B Sub-block

  ≅ Each cache entry will have: 32/8 = 4 valid bits

Write miss: only the bytes in that sub-block is brought in.



Reducing Misses by HW Prefetching Instr, Data

e.g., Instruction Prefetch

≅ Alpha 21064 fetches 2 blocks on a miss

≅ extra block placed in stream buffer

≅ on miss check stream buffer

Could do with data blocks as well:

≅ Jouppi [1990] 1 data stream buffer got 25% misses from 4KB cache; 4 streams got 43%

≅ Palacharla & Kessler [1994] for scientific programs for 8 streams got 50% to 70% of misses from 2 64KB, 4-way associative caches

Prefetching relies on extra memory bandwidth that can use without penalty

Reducing Misses by SW Prefetching Data

Data Prefetch

≅ Load data into Register (HP PA-RISC loads)

≅ Cache Prefetch: load into cache (MIPS IV, PowerPC,SPARC)

≅ Non-faulting (like speculation)

Issuing Prefetch Instructions takes time

≅ Is cost of prefetch issues < savings in reduced misses?

Reducing Misses by Compiler Optimizations

Instructions

≅ Reorder procedures in memory so as to reduce misses

≅ Profiling to look at conflicts

≅ McFarling [1989] reduced caches misses by 75% on 8KB direct mapped cache with 4 byte blocks

Data

@*Merging Arrays*: improve spatial locality by single array of compound elements vs. 2 arrays

@*Loop Interchange*: change nesting of loops to access data in order stored in memory

@*Loop Fusion*: Combine 2 independent loops that have same looping and some variables overlap

@*Blocking*: Improve temporal locality by accessing ϶blocks of data repeatedly vs. going down whole columns or rows

Blocking Example

```
/* Before */
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
      {r = 0;
       for (k = 0; k < N; k = k+1){
          r = r + y[i][k]*z[k][j];};
       x[i][j] = r;
      };
```

2 inner loops:

- ≅ read all NxN elements of z[]
- ≅ read N elements of 1 row of y[] repeatedly
- ≅ write N elements of 1 column of x[]

Capacity Misses a function of N & Cache Size:

- ≅ 3 NxN => no capacity misses; otherwise ...

Idea: compute on BxB submatrix that fits

Blocking Example

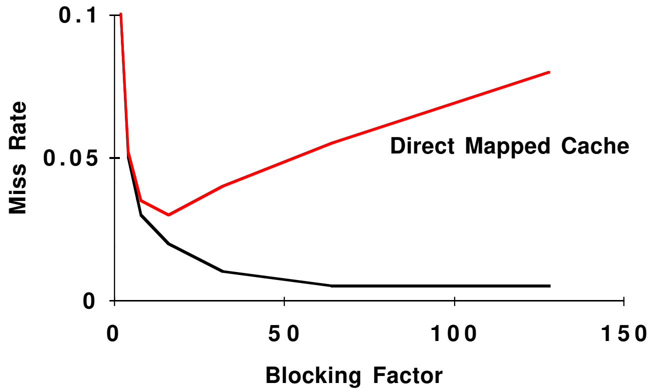
```
/* After */
  for (jj = 0; jj < N; jj = jj+B)
  for (kk = 0; kk < N; kk = kk+B)
  for (i = 0; i < N; i = i+1)
      for (j = jj; j < min(jj+B-1,N); j = j+1)
       {r = 0;
        for (k = kk; k < min(kk+B-1,N); k = k+1)
  {
            r = r + y[i][k]*z[k][j];};
        x[i][j] = x[i][j] + r;
       };
```

Capacity Misses from $N^3 + N^2$ to $N^3/B + N^2$

B called *Blocking Factor*

Conflict Misses Too?

Reducing Conflict Misses by Blocking



Conflict misses in caches not fully associative vs. Blocking size. Lam et al [1991] a blocking factor of 24 had a fifth the misses vs. 48 despite both fit in cache.

Summary of Compiler Optimizations to Reduce Cache Misses